Ciências ULisboa | Informática

Aplicações Distribuídas

2021/22

Projeto 2

1. Descrição geral

A avaliação da disciplina de Aplicações Distribuídas está divida em quatro projetos. O Projeto 2 é uma continuação do Projeto 1, onde os alunos vão resolver três limitações relacionadas com a organização, desempenho e fiabilidade do mesmo. Para além disso, o Projeto 2 é uma oportunidade de os alunos resolverem algum problema pendente do Projeto 1.

O objetivo geral do projeto será concretizar um gestor de bloqueios a recursos para leituras e escritas (*Read-Write Locks*). O seu propósito é controlar o acesso a um conjunto de recursos partilhados num sistema distribuído, onde diferentes clientes podem requerer o acesso aos mesmos de forma concorrente. Um recurso pode ser bloqueado para a escrita exclusivamente por um só cliente de cada vez, até um máximo de **`K`** bloqueios de escrita ao longo do tempo. Ou seja, findo os **`K`** bloqueios de escrita permitidos, o recurso fica desabilitado (i.e., indisponível para novos bloqueios). Um recurso pode ser bloqueado para a leitura por um número qualquer de clientes, caso o recurso não esteja bloqueado para a escrita ou desabilitado. Basicamente, o gestor previne conflitos entre escritas de clientes diferentes, assim como previne conflitos entre operações de leitura e escrita enquanto o recurso está ativo. O gestor será concretizado num servidor escrito na linguagem *Python 3*.

2. Modificações a aspetos definidos no enunciado do Projeto 1

A primeira tarefa consiste em efetuar algumas alterações ao Projeto 1. Neste sentido, no Projeto 2 a comunicação será obrigatoriamente serializada (**ver a PL02 sobre serialização**) e as mensagens trocadas entre o cliente e o servidor seguirão o formato apresentado na Tabela 1, utilizando listas do *Python*. O cliente envia uma lista contendo o código da operação que pretende que o servidor realize, bem como os parâmetros da mesma. Em resposta o servidor enviará também uma lista, com um código de resposta de operação que será sempre o código enviado pelo cliente acrescido de uma unidade. Além deste, o servidor enviará um valor de resultado que substitui as *strings* do projeto anterior.

Tabela 1 - Lista de comandos suportados e formato das mensagens de pedido e resposta.

Comando	Comando recebido pelo cliente	Formato da lista enviada pelo cliente	Resposta do servidor
`LOCK`	`LOCK <R|W> <número do recurso> <limite de tempo>`	`[10, <R|W>, <número do recurso>, <limite de tempo>, <id do cliente>]`	`[11, True]` ou `[11, False]` ou `[11, None]`
`UNLOCK`	`UNLOCK <R|W> <número do recurso>`	`[20, <R|W>, <número do recurso>, <id do cliente>]`	`[21, True]` ou `[21, False]` ou `[21, None]`
`STATUS`	`STATUS <número do recurso>`	`[30, <número do recurso>]`	`[31, 'LOCKED-W']` ou `[31, 'LOCKED-R']` ou `[31, 'UNLOCKED']` ou `[31, 'DISABLED']` ou `[31, None]`

STATS	STATS K <número do recurso>	[40, <número do recurso>]	[41, <número de bloqueios de escrita feitos no recurso>] ou [41, None]
	STATS N	[50]	[51, <número de recursos UNLOCKED>]
	STATS D	[60]	[61, <número de recursos DISABLED>]
PRINT	PRINT	[70]	[71, <estado de todos os recursos>]

Note que em todas as respostas do servidor, onde era enviado um "OK" no Projeto 1 agora é enviado `True`, onde era "NOK" agora é `False` e onde era "UNKNOWN RESOURCE" agora é `None`. Por exemplo, se um cliente com um identificador `15` receber do utilizador o comando "LOCK R 20 100", o programa cliente enviará a lista `[10, R, 20, 100, 15]` e o servidor responderá `[11, None]` se o recurso `20` não existir.

Convém relembrar que o id do cliente deverá ser um número inteiro a definir para cada cliente (passado como argumento na linha de comandos) e que o número do recurso será outro número inteiro entre 1 e `N`, sendo **`N`** o número de recursos geridos pelo servidor de *Locks*. Também, convém relembrar que um dado recurso será bloqueado para escrita até **`K`** vezes, ficando indisponível a partir daí.

Além de serializar e seguir o formato das mensagens, os programas cliente e servidor serão reorganizados segundo o modelo de comunicação baseado em RPC (**ver a PL03 sobre RPC**). Assim, além dos ficheiros atuais, neste projeto teremos os ficheiros `lock_stub.py` (contendo o *stub*) do lado do cliente e `lock_skel.py` (contendo o *skeleton*) do lado do servidor. No servidor, o ficheiro atual será desdobrado em dois: `lock_server.py` (já existente) e `lock_pool.py` (contendo as definições respeitantes ao conjunto de recursos). No cliente, o ficheiro `net_client.py` conterá as definições respeitantes à comunicação do cliente com o servidor, podendo utilizar alguns métodos do ficheiro `sock_utils.py`. A reorganização está ilustrada na Figura 1 (o ficheiro atual `sock_utils.py` não é ilustrado na figura, mas pertence ao projeto).

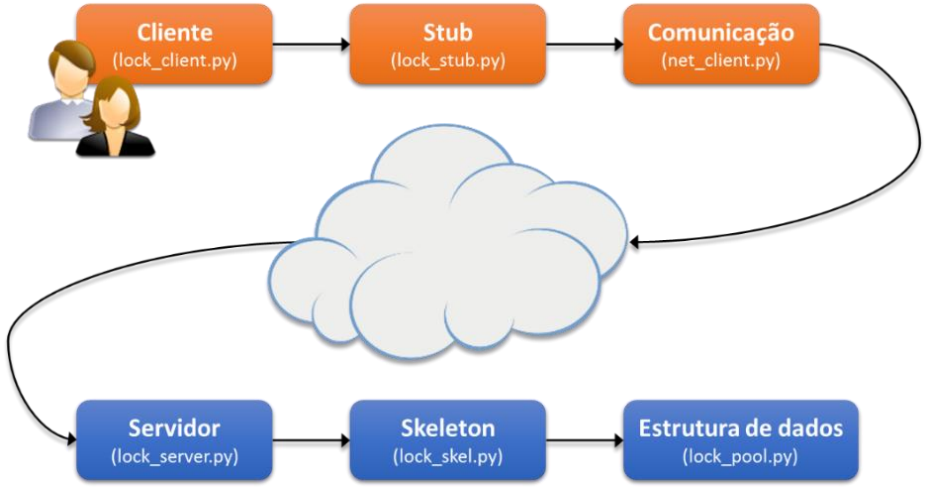

Figura 1 – Reorganização dos programas cliente e servidor.

3. Desempenho: suporte a múltiplas ligações

Nesta segunda fase, o código do projeto deve ser modificado para suportar múltiplos clientes com ligações estabelecidas com o servidor. Para este fim, deve-se usar o módulo *select*, (**ver as aulas TP03 e PL04 sobre suporte a múltiplos clientes com o select**). É de notar que, ao invés do que aconteceu no Projeto 1, pretende-se que os clientes estabeleçam a ligação ao servidor e que esta se mantenha aberta até que o programa cliente seja terminado. Durante a ligação, o cliente poderá enviar múltiplos pedidos.

4. Fiabilidade: tratamento de erros e mensagens parciais

Nesta segunda fase do projeto, os alunos devem certificar-se de que os seus servidores não falham por problemas nos clientes. Além disso, os clientes devem falhar de forma "organizada", *i.e.*, devem mostrar uma mensagem de erro legível e não "rebentar" com mensagens pouco inteligíveis para um utilizador leigo.

Mais concretamente, nesta fase do projeto os alunos devem escrever código para:

1) **Tratar os possíveis erros em todas as chamadas ao sistema.** Isso pode ser feito através do uso de declarações *try/except* para lidar com condições anormais nos programas e realizar as ações para tratá-las de forma limpa.

2) **Ser capaz de receber mensagens fragmentadas.** Durante o Projeto 1 assumimos que a função `socket.recv(L)` devolve o número de bytes `L` que estamos à espera ou uma mensagem completa (caso ela tenha menos de `L` bytes). No entanto, esta função pode retornar menos bytes do que `L`, e portanto, para recebermos uma mensagem completa temos de invocá-la várias vezes até recebermos a mensagem com o tamanho pretendido (**ver aulas TP01 e PL01 sobre *sockets* em *Python* e TP02 e PL02 sobre serialização**). Os alunos devem concretizar no ficheiro `sock_utils.py` a função "*receive_all*" e usá-la no programa em substituição de *recv* (que apenas será usada na concretização de *receive_all*).

5. Validação: validação das mensagens recebidas e enviadas

Nesta segunda fase do projeto, os alunos devem certificar-se de que as mensagens introduzidas pelos clientes sejam validadas por estes antes do seu envio ao servidor, devolvendo uma mensagem de erro. Também, as mensagens recebidas pelo servidor devem ser validadas.

6. Entrega

A entrega do Projeto 2 consiste em colocar todos os ficheiros `.py` do projeto numa diretoria cujo nome deve seguir exatamente o padrão `grupoXX` (onde `XX` é o número do grupo). Juntamente com os ficheiros `.py` deverá ser enviado um ficheiro de texto `README.txt` (é em TXT, não é PDF, RTF, DOC, DOCX, etc.) onde os alunos podem relatar a informação que acharem pertinente sobre a sua implementação do projeto (por exemplo, limitações). A diretoria será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão `grupoXX.zip` (novamente `XX` é o número do grupo). Esse ficheiro será submetido num recurso a disponibilizar para o efeito na página de AD no moodle da FCUL. Note que a entrega deve conter apenas os ficheiros `.py` e o ficheiro `README.txt`, qualquer outro ficheiro vai ser ignorado. O não cumprimento destas regras podem anular a avaliação do trabalho.

O prazo de entrega do Projeto 2 é **domingo, 3 de Abril de 2022, às 23:59 (WEST)**.

7. Plágio

Não é permitido aos alunos dos grupos partilharem código com soluções, ainda que parciais, a nenhuma parte do projeto com alunos de outros grupos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um sistema verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso poderá ser reportado aos órgãos responsáveis na Ciências@ULisboa.

8. Referências

- https://docs.python.org/3/tutorial/classes.html
- https://docs.python.org/3/library/socket.html
- https://docs.python.org/3/library/exceptions.html
- https://docs.python.org/3/library/time.html
- https://docs.python.org/3/library/select.html
- https://docs.python.org/3/library/pickle.html